Failure Modes and Effect Analysis of Use Cases: A Structured Approach to Engineering Fault Tolerance Requirements

Elena Troubitsyna
Åbo Akademi University, Department of Computer Science
Joukhaisenkatu 3-5A, 20520, Turku, Finland
Elena.Troubitsyna@abo.fi

***Abstract*— Fault tolerance – an ability of a system to cope with errors – is an important characteristic of dependable systems. However, software development approaches traditionally give precedence to modelling nominal system behaviour over modelling system behaviour in presence of faults. This leads to ad-hoc and error prone implementation of fault tolerance mechanisms. In this paper, we propose a systematic approach to elicitation and modelling of fault tolerance-related requirements. Our approach is based on using Failure Modes and Effect Analysis (FMEA) that is used to identify faults, their detection and error recovery. We rely on use-case modelling to structure system behaviour and propose to conduct FMEA of each individual use case. Our approach facilitates elicitation and structuring of fault tolerance behaviour. It enables an integrated modelling of nominal and abnormal system behaviour from early development phases.**

Keywords - use cases; failure modes and effect analysis (FMEA); fault tolerance; requirements

I. INTRODUCTION

The model-driven approaches to software development [1] usually represent system functionality in term of use cases. Use cases [2] describe system behaviour at different levels of abstraction. At the highest level of abstraction they depict the services that the system provides to its users. At the lower layers of abstraction, they describe functions of system components. Use-case modelling facilitates structuring complex requirements and serves as a basis for validating system design at the later stages of the development.

Traditionally modelling focuses on describing nominal system functionality. Yet, there are also many abnormal (exceptional) situations that arise during system execution. System dependability [3] can be jeopardized if such abnormal situations are not handled in a proper way, i.e., if fault tolerance mechanisms are implemented incorrectly. Although fault tolerance mechanisms constitute a significant part of software, they are often introduced at the implementation stage and in a rather ad-hoc fashion.

In this paper we propose an approach to conducing failure modes and effect analysis (FMEA) [4] over the use cases. FMEA is a widely used inductive safety analysis technique. We demonstrate how to apply FMEA to represent abnormal situations in use case execution. Our approach allows the designers systematically explore exceptional situations, identify their causes and error recovery strategy. We propose the patterns for conducting FMEA at different levels of abstraction and demonstrate how to incorporate the results of such an analysis into use case representation. The requirements obtained while conducting FMEA are systematically captured in the use case system model.

It is widely accepted that building in dependability and in particular, fault tolerance, early in the development process is more cost-effective and results in more robust design [3,4]. Our approach facilitates early consideration of fault tolerance in the design process. It allows the designers to uncover the additional requirements, which are needed to ensure fault tolerance. Moreover, it makes the process of requirements engineering more structured and hence improves requirements traceability.

The proposed approach is illustrated by a case study – modelling and analysis of an autonomous robot.

II. MODELING FAULT-TOLERANT SYSTEMS

The main goal of introducing fault tolerance is to design a system in such a way that faults of components do not result in a system failure [3,5,6]. A fault manifests itself as *error* – an incorrect system state [3,10]. Nowadays the main part of fault tolerance mechanisms are software implemented, i.e., software should detect errors and initiate error recovery. Error recovery is an attempt to restore a fault-free system state or at least preclude system failure. There are two types of error recovery: dynamic and fail-safe recovery. In the former case, upon detection of error software executes certain actions to restore a fault-free system states and then resumes normal system functioning without stopping the system. In contrast, fail-safe error

 ISBN: 978-1-61208-149-6

recovery brings the system into a safe but non-operational state, i.e., executes system shut-down.

Initially the system is assumed to be fault free. Upon successful initialization, the system enters an automatic operating mode. While no error is detected, the system executes the normal control functions. Upon detection of an error software tries to execute error recovery and resumes normal function. If error is deemed to be fatal then software ceases its function and notifies the operator about it (e.g., by raising an alarm).

Use case modelling is a widely used technique for discovering and representing behavioural requirements of software-intensive systems [2]. A *use case* describes, without revealing the system implementation details, the system responsibilities and interactions with its environment while providing the requested services. A use case represents a distinct unit of interaction between an environment (human or machine) called an *actor* and the system. In general, the actors can be thought of as different stake holders that request the services to pursue their goals. Each use case describes a certain functionality to be designed. It can also include another use cases. Usually a use case contains several scenarios – the main scenario describing successful execution – and an alternative one describing various deviations.

There are ongoing debates on the principles of use case modelling. For instance, Fowler advices against breaking use cases into sub- use cases [7]. We argue that this approach is unsuitable for development of large-scale industrial systems. In this paper we adopt refinement-based approach to use case modelling. Namely, we propose to build the use case model gradually in a top-down manner.

Our initial model describes system functionality in terms of services delivered by the system in response to the service requests. The service requests are generated by the system environment represented by a certain actor. While designing a service, we should provide means for tolerating faults of various natures. In general, an execution of each service can fail. Hence each use case should contain means for fault tolerance. We propose to supplement each use case with an auxiliary use case defining a fault tolerance mechanism, which should be activated if an execution of the main use case fails. The initial use case diagram describes the basic use cases and supplements each of them with the auxiliary use cases to model error recovery. Observe that the auxiliary use cases confine all possible alternative actions to be undertaken for error recovery. In Fig. 1 we propose a general pattern for use-case modelling of a fault-tolerant system at the abstract level.

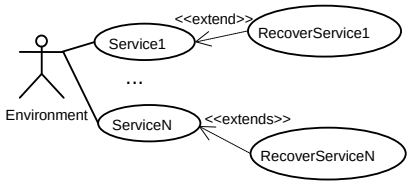

Figure1. Use case diagram of fault-tolerant system

Often at the abstract level of modelling the requirements describing the fault tolerance mechanism are yet to be discovered. However, even an abstract representation of them in the use case diagram shown in Fig. 1 enforces early consideration of fault tolerance aspect and facilitates elicitation of the requirements related to fault tolerance.

Usually a service provided by a system is a composition of certain subservices. In the use case modelling this can be depicted by decomposing the abstract use cases and refining the overall use case model. On the one hand, the refined use case diagram introduces the lower-layer use cases and defines relationships between the use cases on the higher and the lower layers. Such relationships are depicted via the stereotype `<<include>>`, since an execution of the upper-layer use case involves the execution of several lower-layer use cases. On the other hand, the refined use-case diagram specifies more precisely the fault tolerance mechanisms, which should be introduced to provide error recovery at each level of abstraction. Eventually we arrive at the use case diagram of the form presented in Fig. 2.

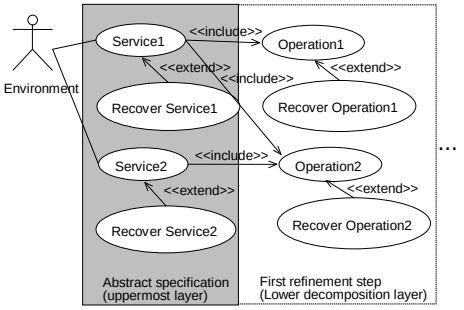

Figure 2. General pattern for final use case diagram

Observe that the use case diagram has the layered structure. The first layer encompasses the use cases describing functionality of the system from the environment perspective. They define the services, which the environment expects from the system. Each refinement step introduces the lower layers, which contain the use cases whose execution is required to provide the use cases at the upper layers. The decomposition is rendered via the `<<include>>` stereotype. The fault tolerance mechanisms are related with the corresponding

 ISBN: 978-1-61208-149-6

use cases via the <<extend>> stereotype, as we discussed previously.

The occurrence of errors might prevent accomplishing the actor's goals. While designing a system it is important to ensure that each service request is acknowledged either by the successful result or by a meaningful error message. An error message might also contain the information that gives the actor a recommendation on how to achieve the desired goal by the other means.

In Fig. 3 we show the general template of use case description of a service as well as error handling use case. The general template can be applied to describe use cases at all levels of abstractions. It is easy to observe that fault tolerance mechanism has a hierarchical structure – failure of lower layer use cases are handled from higher-level recovery use cases.

It is easy to observe that completeness of fault tolerance requirements directly depends on whether our analysis of possible failures modes of use cases is exhaustive. To facilitate the analysis of possible failure modes of use cases, we propose to use Failure Mode and Effect Analysis (FMEA) [4]. It is a well-known inductive technique for eliciting failure modes of system components. Next we demonstrate how to apply this technique to facilitate discovery of failure modes of use cases.

Description of use case *Operation_Name*
Precondition When use case can be executed
Postcondition Normal result
Exceptional result
Includes Lower layer use cases
Normal sequence of events:
1. Check input parameters. If check fail execute use case *Recover_Operation*
2. Steps of use case. If a step includes invocation of lower layer use case then check the response. If check succeeds then proceed. Otherwise invoke use case *Recover_Operation*

Description of use case *Recover_Operation*
Precondition Failed input parameters or included use case execution
Postcondition Handling error
Extends: *Operation_Name*
Sequence of events:
1.Check invocation parameters. In case of input parameters failure, generate corresponding error and abort execution
2.In case of included use case failure apply appropriate recovery actions, e.g., retry, rollback, abort, reconfiguration. If recovery fails, generate corresponding error. Ensure that recovery actions are specified for each possible error.

Figure 3. Template of detailed use case description

III. INTEGRATING FMEA AND USE CASES

FMEA [4] is an inductive analysis method, which allows us to systematically study the causes of components faults, their effects and means to cope with these faults. FMEA is used to assess the effects of each failure mode of a component on the various functions of the system as well as to identify the failure modes significantly affecting dependability of the system. FMEA step-by-step selects the individual components of the system, identifies possible causes of each failure mode, assesses consequences and suggests remedial actions. The results of FMEA are usually represented in the tabular form that contains the following fields: component name, failure mode, possible cause, local effect, system effect, detection, and remedial action.

In this paper we propose to use FMEA to derive possible failure outcomes of each use case execution. To facilitate FMEA of use cases we introduce taxonomy of possible failure modes of use cases and outline corresponding detection procedures and remedial actions. Below we present the corresponding FMEA tables for the typical failure modes.

Use case	Use case name (uppermost layer)
Failure mode	Incorrect input parameters
Possible cause	Human or computational error
Local effects	Use case cannot be executed
System effect	Failure to execute requested service
Detection	Check value of input parameters before starting to execute use case
Remedial action	Abort service execution, return error message to environment

This failure mode represents an attempt to invoke a service with the incorrect input parameters. It is an unrecoverable error. While describing a use case, we should ensure that the returned erroneous service response identifies the causes of the failure.

Use case	Use case name (lower layer)
Failure mode	Incorrect input parameters
Possible cause	Computational error
Local effects	Use case cannot be executed
System effect	Failure to execute requested subservice
Detection	Check value of input parameters before starting to execute use case
Remedial action	Abort use case execution, suspend service provision, return error message to the service requester

This failure mode represents a failure to execute lower layer use case due to incorrect input parameters. Usually occurrence of such a failure would correspond to receiving an exception [8]. As an error recovery, the service requester should diagnose the cause of failure either by re-computing the input parameters or by propagating the exception further in the use case hierarchy.

 ISBN: 978-1-61208-149-6

Use case	Use case name (uppermost layer)
Failure mode	Incorrect service provision (wrong postcondition)
Possible cause	Computational error or unrecoverable error of subservices, or physical component failure
Local effects	Incorrect provision of service
System effect	Service is executed incorrectly
Detection	Check postcondition, generate error message, implement logging
Remedial action	Abort service execution, return error message to environment, halt system

The failure mode described above analyses an occurrence of a failure while executing a service. The failure might be caused by a failure of a lower layer use case or by a computational error at a higher level of abstraction. The diagnostic of such a failure aims at identifying its causes and deciding on the appropriate error recovery strategy.

The use case describing a similar type of failure of lower layer use case can be defined in the same way. In case the failure is transient, the error recovery by retry would possibly bring the system back to the normal state. In case the failure cannot be recovered, the error message is propagate to the upper layers of hierarchy.

The use case below describes service omission error. It is detected by the missed deadline. The failure mode of the use case on the lower layer is defined in the similar way.

Use case	Use case name (uppermost layer)
Failure mode	No response
Possible cause	Computational error or failure of lower layer use cases or communication failure
Local effects	Use case is not executed
System effect	No response on service requests within the deadline
Detection	Timeout
Remedial action	Execute system diagnostics. If no error is detected then resume normal operation otherwise halt system

Let us demonstrate the application of the proposed method to structure and model requirements of an autonomic robot.

IV. CASE STUDY

We illustrate use-case modelling of a fault-tolerant system by an example – *an autonomic robot.* The robot should move on a surface, i.e., in XY- directions and grab the objects located at certain positions. Via a radio-link a human operator sends the robot commands to move from one position to another, grab and release objects. The robot works autonomously. Such kind of robots are used in the environments that are hazardous for humans, e.g., to perform rescue operations. Since faults might prevent the robot from executing the requested service (that might lead to a failure of the rescue operation), the system has strict fault tolerance requirements.

The service-level use case model of the robot shown in Fig. 4 is very simple. It has two main use cases – move to the target coordinates and grab/release – and two auxiliary use cases to implement error recovery.

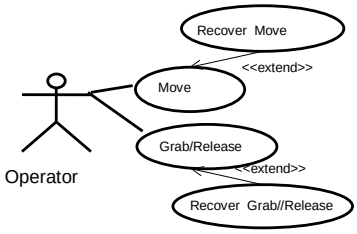

Figure 4. Service-level use-case model of the robot

Let us demonstrate how FMEA of the services-level use cases facilitates elicitation of fault tolerance requirements.

Use case	Move
Failure mode	No response
Possible cause	- Communication failure - Lack of mechanism to detect timeout of lower layer use cases - Computational error (non-termination)
Local effects	Use case is not executed
System effect	No response on service requests within the deadline
Detection	Timeout
Remedial action	- Retry execution of the use case. If execution succeeds then resume normal operation. - To diagnose communication failure send ping request. If no response then halt the system. - To ensure that execution terminates, set deadlines for execution of each lower layer use case. - To ensure termination guarantee termination of error recovery and proper handling of exceptions.

The example of FMEA allows us to identify important requirements, such as *introduce timers, ensure termination of error recovery and additional functionality required to implement diagnostics of communication failure.* Moreover, the system design should also ensure that *upon completing each service, the success or failure of the execution is checked.*

The service-level use case diagram is further refined to model the details of use-case implementation as shown in Fig. 5. Each service is decomposed into the lower layer use cases, which should be executed to implement it.

 ISBN: 978-1-61208-149-6

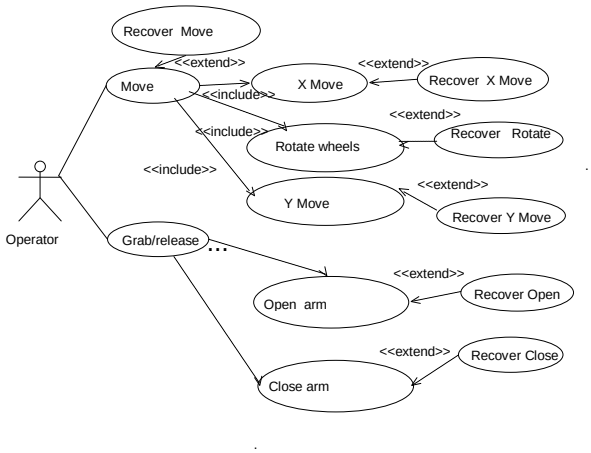

Figure 5. Refinement of use-case diagram of the robot

The diagram is created according to the pattern proposed in Fig. 2. The main use cases are decomposed into the lower layer use cases, which should be executed to implement them. At this refinement step, a more detailed description of error recovery can be given as well. We add the details of the possible causes of faults, detection and error recovery into our FMEA analysis. This allows us to arrive at the detailed description of use cases. Below we present a detailed description of the *Move* use case, which contains description of the possible errors and corresponding error recovery procedures derived from FMEA results.

The precondition might fail, if the operator inputs wrong parameters. In the use case *Recover Move* we model the notification of the operator and shut down. The execution of *Move* essentially consists of executing the included lower-layer use cases *Move to X position*, *Rotate Wheels* and *Move to Y position*. The recovery from failures is done by executing *Recover Move* with the parameters corresponding to the names of the failed use case.

At the final refinement step the details of executing the lowest-layer use cases and failure modes of the components, involved in the execution become available. This allows us to establish the exact causes of failures of the lowest-layer use cases and precisely define the required remedial action precisely. We omit presenting the detailed description of the lower layer use cases. The use case diagram obtained at the final step adheres to the pattern presented in Fig. 4. The detailed description of functional requirements and the requirements related to fault tolerance are obtained at this stage.

In this section we have illustrated the evolution of use-case model of a fault-tolerant system. We emphasized reasoning about fault tolerance at each refinement step.

In general the proposed approach can be seen as a generalization of Randell's recovery block approach [6]

Use case Move

Brief description This use case defines system reaction on the operator's command **"move to XY coordinate"**. It includes activating motor, rotating wheels, reading positioning sensors, reporting success or failure of the execution

Includes use cases **"***X Move", "Y Move", "Rotate Wheels"*

Extends use case *"Recover Move"*

Preconditions Operator requests service **Move to X,Y position,** the system is fault free

Postconditions The robot reaches the requested position before the deadline and success is reported. Otherwise failure is reported

Normal sequence of events

1. Verify that *X,Y* are valid coordinates. If the verification fails then **A_Failure1** in recovery sequence, else calculate the distance along the X direction and Y direction.
2. Execute the use cases *"Move to X position"*
3. If the execution of the use cases *"Move to X position"*, failed then **A_Failure 2** in recovery sequence, else proceed to execution of the use cases *"Rotate Wheels"*
4. Execute *"Rotate Wheels"*. If execution of *"Rotate Wheels"* failed then **A_Failure 3** otherwise proceed to execution of *"Move to Y position"*
5. If the use case *"Move to Y position"* failed then **A_Failure 4** in recovery sequence, else if execution of the use case succeeded then report the success of the service execution
5. **…**

Recovery sequence of events

A_ Failure1: Execute the use case *"Recover Move"* with the parameter "incorrect input parameters**".** Shut-down the system. Notify the operator

A_Failure2: If the use case *"Move to X position"* has failed then execute the use case *"Recover Aspirate"* with the parameter *"Move to X position"*. If the use case *"Move to Y position"* has failed then execute the use case *"Recover Aspirate"* with the parameter *"Move to Y position"*.

…

and its translation into use-case modelling. Indeed, a recovery block describes how the operator can achieve a desired goal in the normal as well as erroneous situations. The recovery block has a flat structure, i.e., the operations required to achieve the desired goal reside on the same level of abstraction. Complexity of modern systems requires an encapsulation of the low-level operations and hence, imposes the hierarchical style of system structuring. In our approach we generalized the recovery block mechanism by introducing the hierarchy of use cases defining how the desired goal can be achieved.

 ISBN: 978-1-61208-149-6

V. CONCLUSIONS

In this paper we demonstrated how to structure complex requirements by FMEA and refinement of use cases. We propose a pattern for use-case model of fault-tolerant systems and demonstrated how to structure the description of use cases to capture the fault tolerance aspect. The pattern aims at enforcing early consideration of fault tolerance in the design process. It allows the designers to uncover the additional requirements, which are needed to ensure fault tolerance. Moreover, the proposed pattern makes the process of requirements engineering more structured and hence improves requirements traceability.

Among the pioneering works on addressing dependability in UML modelling is research by Alexander on misuse cases [1]. He proposed to consider use cases with hostile intent to facilitate discovery of dependability-related requirements. While the developers of safety- and security-critical systems are familiar with misuse-case modelling, the developers of non-critical systems traditionally rely on use-case modelling. Often safety or security implications are unknown in the beginning of the system development and uncovered later in the development process. As a result, the development of a critical system is conducted in a traditional style. Our approach aims at addressing this problem by enabling dependability consideration in the traditional UML modelling.

Alenby and Kelly [9] studied use-case modelling as a tool for discovering safety-related requirements. Our approach is similar to their work, since the new requirements can be discovered as well. However, our main focus was to study how to systematically capture requirements by conducting FMEA.

Kelly and Weawer [10] proposed an extension of goal structuring notation (GSN) to support safety argument. They demonstrated how GSN can facilitate safety assurance via structuring safety case. Our approach employs the similar idea of decomposing the high-level goals into the low-level sub-goals. However, we focused on the development process rather than on description of safety cases.

Jurjens focused on algebraic formalization of various UML artefacts to reason about safety [11]. However, his work leaves aside the problem of capturing dependability-related requirements in the development process.

Use cases have been formalized as contracts by Back et al. [12]. However, this work does not consider dependability aspects.

Hassan et al. [13] proposed a methodology that enables architectural-level analysis of safety using a combination of safety techniques. Our approach provides a support for early development stages and hence can be considered as complementary to [13]. The opposite approach – deriving FMEA from UML models has been explored by David et al [14]. The similar issues have been studied by Mazzara is the problem frames [15]. The use of FMEA at different stages of UML-based development has been explored by Hecht et al [16] and Wentao [17].

In our future work we are planning to explore further various fault tolerance mechanisms and their modelling in UML.

REFERENCES

[1] I. Alexander. Misuse cases: Use Cases with Hostile Intent. *IEEE Software, vol.20 (1),* pp. 58-66, 2003.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. "*The Unifying Modeling Language User Guide*". Addison-Wesley, 1999.

[3] J.-C. Laprie. Dependability: Basic Concepts and Terminology. Springer-Verlag, Vienna, 1991.

[4] N. Storey. *Safety-critical computer systems*. Addison-Wesley, 1996.

[5] T. Anderson and P.A. Lee. *Fault Tolerance: Principles and Practice.* Dependable Computing and Fault-Tolerant Systems, Vol 3**.** Springer Verlag; 1990.

[6] B. Randell and J. Xu, The Evolution of the Recovery Block Concept. In M. Lyu (ed.) *Software Fault Tolerance*. Wiley 1994.

[7] M.Fowler. UML Distilled: A brief Guide to the Standard Object Modelling Language. Addison-Wesley, 2004.

[8] F. Cristian. Exception Handling. In T. Anderson (ed.): *Dependability of Resilient Computers*. BSP Professional Books, 1989.

[9] K. Allenby, T. P. Kelly. Deriving Safety Requirements using Scenarios. In *Proc. of the 5th IEEE International Symposium on Requirements Engineering (RE'01),* Toronto, Canada, pp.228-235, 2001.

[10] T. Kelly and R. Weaver. The goal Structuring Notation – A Safety Argument Notation. In Proc. of *The Dependable Systems and Networks 2004 Workshop on Assurance Cases*, July 2004.

[11] J. Jürjens. Developing safety-critical systems with UML. In *Proc. of UML'2003***.** Lecture Notes in Computer Science, San Francisco, USA, October 2003, pp**.**360 – 372.

[12] R.-J. Back, L. Petre and I. Porres. *Analysing UML Use Cases as Contracts.* In Proceedings of UML'99. Fort Collins, Colorado, USA, October 1999. Lecture Notes in Computer Science 1723, pp. 518-533, Springer-Verlag.

[13] A.Hassan, K. Goseva-Popstojanova, K and H. Ammar. UML based severity analysis methodology. In Proc. of Reliability and Maintainability Symposium. Computer Press, 2005

[14] P. David, V. Idasiak & F. Kratz. Towards a better interaction between design and dependability analysis: FMEA derived from UML/SysML models. In Proc of ESREL 2008 and 17th SRA-EUROPE, Spain, 2008.

[15] M. Mazzara, Deriving Specifications of Dependable Systems: toward a method. CS-TR 1152 –Technical report Newcastle University, May 2009.

[16] H.Hecht, X.An and M.Hecht. Computer-Aided Software FMEA. In Proc. of RAMS 2004, Computer Press, 2004.

[17] W.Wentao and Z.Hong. FMEA for UML-Based Software. In Proc. of World Congress on Software Engineering, Computer Press, 2009.

 ISBN: 978-1-61208-149-6